Natural Deduction Proof Checker
User's Guide

Brian Mulanda and Rod Howell
Dept. of Computing and Information Sciences
Kansas State University
Manhattan, KS
USA

Version 0.2
January 30, 2007

Contents

1 Introduction

This User's Guide documents the installation and use of the Natural Deduction Proof Checker, a program for checking the correctness of natural deduction proofs. This is not a tutorial on natural deduction proofs. The style of proofs acceptable to this program is patterned closely after the presentation given in [1], which gives an excellent introduction to natural deduction proofs.

Please submit all bug reports, including errors in this document, to

rhowell@ksu.edu

Include the following information regarding any software bug:

- A description of the cause of the error. If possible, provide enough information so that the bug can be reproduced. If relevant, attach a proof illustrating the bug.
- An explanation of why you believe this to be a bug (unless this is obvious).
- The operating system you are using.
- The version of the Java™ Runtime Environment your machine is using (if you know it).

The remainder of this document is organized as follows. Section 2 gives instructions for installing the program. Section 3 describes the basic interactions with the program, without going into the details of how correct proofs are constructed. Section 4 then describes how propositional logic proofs are constructed and checked. Section 5 extends this description to predicate logic proofs.

2 Installation

The Natural Deduction Proof Checker is currently available only to students of CIS 301 at Kansas State University. The program should run on any platform with a graphical user interface and the Java Runtime Environment (JRE), version 1.4 or later, installed. (The program has not been tested under earlier versions of the JRE.)

If you are enrolled in CIS 301 at KSU, you may download the file ndpc.jar from K-State Online to your desktop or another convenient location. Opening this file (typically by double-clicking on its icon) will start the program. Alternatively, the program can be started from a command line using the following command:

```
java -jar ndpc.jar
```

If the program does not start, or if certain functionality appears to be absent, it is likely that an appropriate version of the JRE is not installed on your machine. To install the latest JRE, open

http://java.sun.com/javase/downloads/index.jsp

in a web browser, and click on the button labeled "Download" next to "Java Runtime Environment". Follow the instructions on the resulting page. If you are installing the JRE on a Microsoft Windows operating system, you should install the Offline Installation.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

If you wish to install an earlier version of the JRE, open

http://java.sun.com/downloads/

and select your desired version of J2SE from the drop-down list entitled, "Full Java SE Technology Downloads List". On the resulting page, click the link entitled, "Download J2SE JRE", and follow the instruction on the resulting page. Again, for a Microsoft Windows operating system, use the Offline Installation.

3 The User Interface

The proof checker graphical user interface is composed of two main parts: the proof area and the message area (see Figure 1). The proof area is where you would key in the proof. This area will always be blank when you start the program. The message area, located at the bottom of the interface, is where messages — errors or information — are displayed during use of the program. There is a movable divider between the proof area and the message area. This divider can be move up or down by clicking and dragging with the mouse. Also, clicking on the two triangles on the left-hand side of the divider will cause the proof area or the message area to be hidden or revealed.

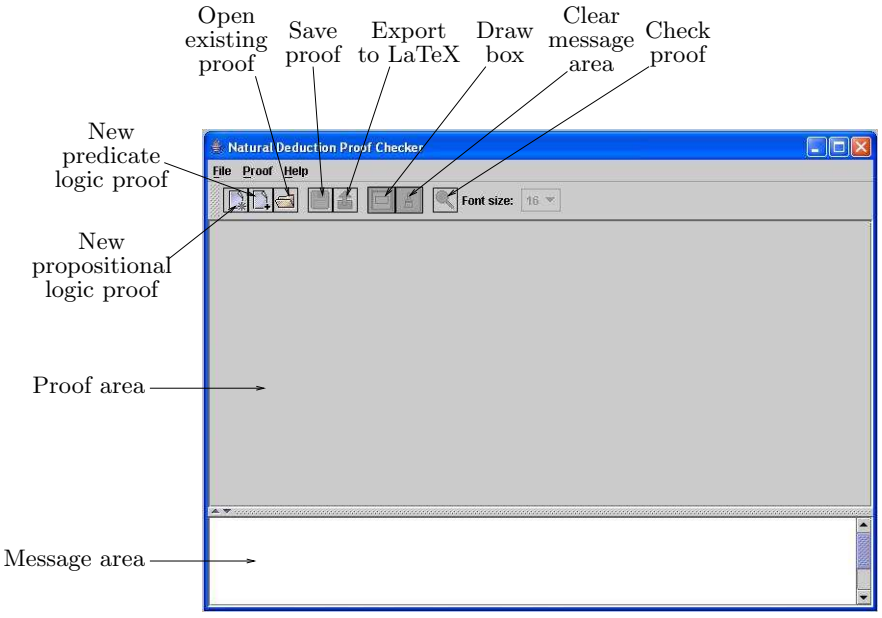

Figure 1: The initial program window.

The remainder of this section describes all of the actions that can be taken from this window. Because there is no proof in the window initially, only three of these actions are enabled at this time:

- **New Propositional Logic proof** (Section 3.1);
- **New Predicate Logic proof** (Section 3.2); and
- **Open existing proof** (Section 3.10).

Once one of these actions is taken to begin or resume a proof, the remaining actions are enabled.

3.1 Starting a New Propositional Logic Proof

A new propositional logic proof may be started in any of the following ways:

- Using the File menu: Select New, then Propositional Logic proof.

- Using pop-up menu: Right-click in the proof area, then select New, then Propositional Logic proof.
- Using toolbar: Click the button shown in Figure 1.
- Using the shortcut key `Ctrl+P`.

If the proof area contains a proof that has not been saved, a confirmation dialog is displayed asking whether the proof should be saved. If you click Yes, the proof will be saved as described in Section 3.8. If you click No, the old proof will not be saved before the new proof is started. If you click Cancel or close the dialog box, a new proof will not be started, and the old proof will not be saved.

Starting a new propositional logic proof causes the proof area to change, as shown in Figure 2. In addition, the title of the window now gives the following information:

- Propositional Logic proof: Indicates the type of logic you are using.
- Untitled: Indicates that no file is associated with this proof. Saving the proof (see Section 3.8) will cause the file name to be displayed here instead.
- The * at the end indicates that the proof has not been saved. Saving the proof (see Section 3.8) will cause the * to disappear.

3.2 Starting a New Predicate Logic Proof

A new predicate logic proof may be started in any of the following ways:

- Using the File menu: Select New, then Predicate Logic proof.
- Using pop-up menu: Right-click in the proof area, then select New, then Predicate Logic proof.
- Using toolbar: Click the button shown in Figure 1.
- Using the shortcut key `Ctrl+D`.

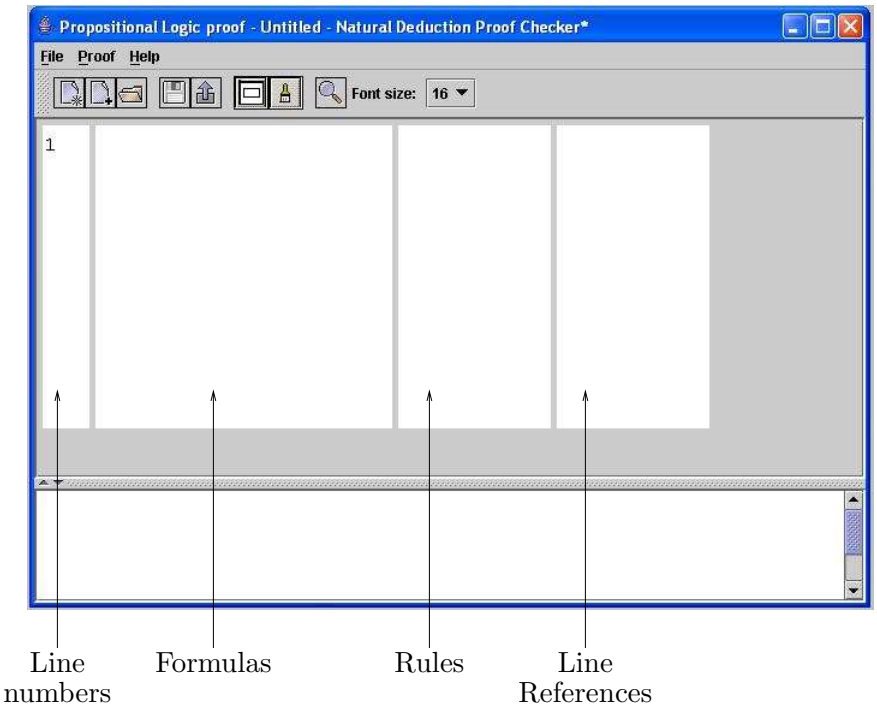

Figure 2: A new propositional logic proof.

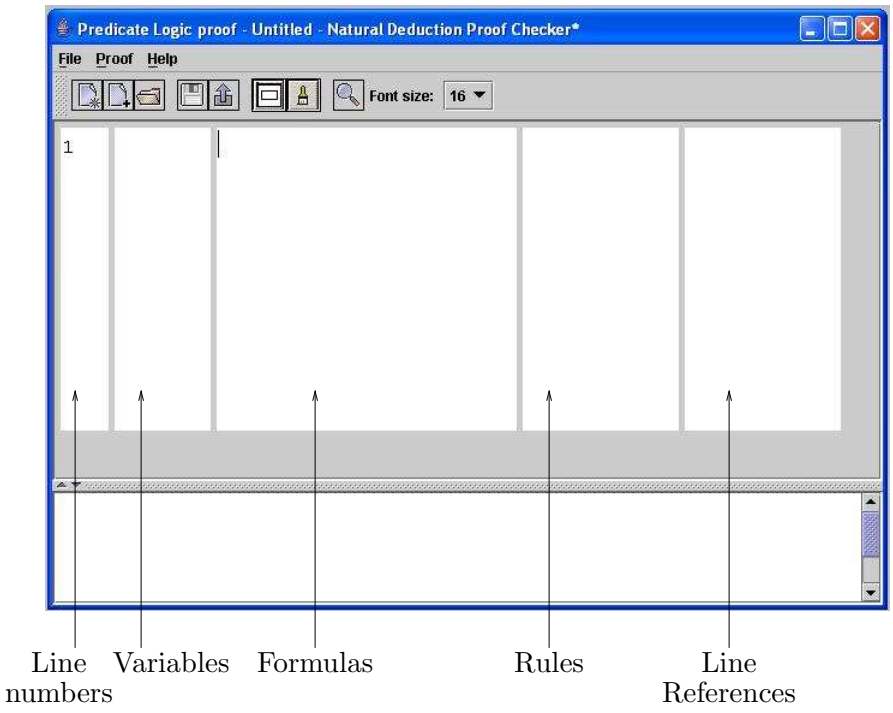

Figure 3: A new predicate logic proof.

If the proof area contains a proof that has not been saved, a confirmation dialog is displayed asking whether the proof should be saved. If you click Yes, the proof will be saved as described in Section 3.8. If you click No, the old proof will not be saved before the new proof is started. If you click Cancel or close the dialog box, a new proof will not be started, and the old proof will not be saved.

Starting a new predicate logic proof causes the proof area to change, as shown in Figure 3. The window title also contains information about the proof, as described in Section 3.1.

3.3 Editing a Proof

Text may be typed into any of the columns of the proof area except the first (i.e., the line numbers column) in order to construct a proof. In most cases the text will appear exactly as typed. There are three characters, however,

that will be displayed differently from their appearance on the keyboard:

Key	Appearance	Usage
Upper-case `A`	$\forall$	Universal quantifier (predicate logic)
Upper-case `E`	$\exists$	Existential quantifier (predicate logic)
Underscore (_)	$\bot$	Contradiction, or "bottom"

See Sections 4 and 5 for details on how valid proofs are constructed.

You may move the text cursor from one position to another in the proof using either the keyboard or the mouse. Using the mouse, you may click on any of the editable columns of an existing line to position the text cursor within that column and on the chosen row. Keyboard navigation is described in what follows.

3.3.1 Navigating the columns in a proof using the `Tab` key

Pressing the `Tab` key only moves the cursor to the next column to the right. If the cursor is currently on the last (extreme right) column, pressing the `Tab` key moves the cursor to the first editable column of the next row down. The `Tab` key will have no effect if the cursor is on the last column of the last row in the proof.

Pressing `Shift+Tab` moves the cursor to the previous column to the left. If the cursor is currently on the first editable column, pressing the `Tab` key moves the cursor to the last column of the previous row up. `Shift+Tab` will have no effect if the cursor is on the first editable column of the first row in the proof.

3.3.2 Navigating with the arrow keys

Normally, the left and right arrow keys will move the text cursor one character to the left or right, respectively. When the text cursor is positioned after the last character in a column, pressing the right arrow key will have the same behavior as pressing the `Tab` key. When the text cursor is positioned at the beginning of a column, pressing the left arrow key will have the same behavior as pressing `Shift+Tab`.

To move to the next row down, press the down arrow key. To move to the previous row up, press the up arrow key.

3.3.3 Adding new lines to the proof

You add a new line in the proof by pressing the `Enter` key. Any text to the right of the cursor at the time you press the `Enter` key will be moved to the next row down. For example, to add a new blank line after a row, position the text cursor to the right of all text on that row. Then press `Enter`. Alternatively, to add a new blank line prior to a row, position the text cursor at the beginning of the first editable column in that row. Then press `Enter`.

Whenever a new line is added, it is given a line number in the first column, and the line numbers of all succeeding lines are incremented by 1. In the last column, line numbers referring to lines following the line containing the cursor are incremented by 1. Note that this means that if the cursor is placed at the beginning of line i and `Enter` is pressed, references to line i in the last column will *not* be incremented. For this reason, it usually works better to insert new lines by placing the cursor at the end of the previous line whenever possible.

In both propositional and predicate logic proofs, when a new line is added, the cursor is placed in the formula column.

3.3.4 Deleting lines from the proof

Only blank lines can be deleted from the proof. You use the `Backspace` or `Delete` keys to delete text from lines in the proof. (Pressing the `Backspace` key at the beginning of a column in a non-blank row has the same effect as pressing `Shift+Tab`; likewise, pressing `Delete` at the end of a column in a non-blank row has the same effect as pressing `Tab`.) You can then delete the blank line as follows.

To delete a blank line using the `Backspace` key, position the text cursor in the first editable column of that line. Ensure that the line does not contain any text. Then press the `Backspace` key. The blank line will be removed and the text cursor will be positioned at the end of the previous row.

If the line to be removed is the first line in the proof, using the `Backspace` key will not delete the line. Instead use the `Delete` key, as follows.

To delete a blank line using the `Delete` key, position the text cursor in the last column of that line. Ensure that the line does not contain any text. Then press the `Delete` key. The blank line will be removed and the text

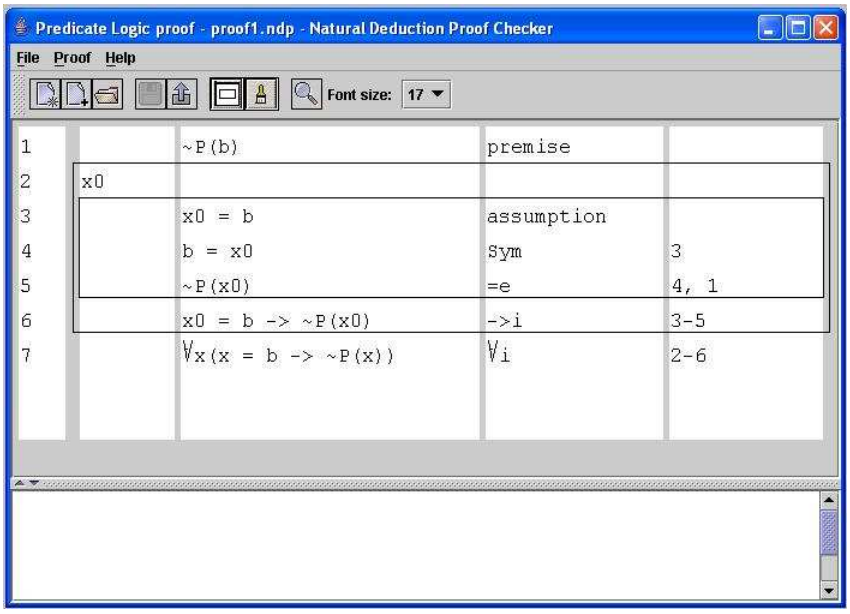

Figure 4: A proof of `~P(b)` $\vdash \forall$ `x(x = b -> ~P(x))`.

cursor will be positioned at the beginning of the next row, which by now is the current row.

If the line to be removed is the last line in the proof, using the `Delete` key will not delete the line. Instead use the `Backspace` key.

Whenever a line is deleted, the line numbers in the first column for all succeeding lines are decremented by 1. Line numbers in the last column are also updated accordingly. Any references in the last column to the deleted line are replaced by a question mark (`?`).

3.4 Using Proof Boxes

Boxes are an important part of most propositional and predicate logic proofs (see, e.g., Figure 4). This section describes how these boxes may be drawn, deleted, and modified. Refer to Sections 4 and 5 for details on the correct use of boxes in proofs.

3.4.1 Valid boxes

Boxes may not overlap; i.e., they may be nested one completely inside of another or completely separate, but one box cannot be both partly inside and partly outside another box. If any of the operations described in this section would result in an invalid box, that operation will fail and cause an error message to be displayed in the message area.

3.4.2 Drawing a box

You can draw boxes in any of the ways described below.

- Using the mouse: First, click and hold down the left mouse button anywhere within the proof area. Then drag the mouse upward or downward and release. As you drag the mouse, the user interface provides visual feedback of which rows will be enclosed by the box when you release the mouse. To abort the drawing of the box, press `Esc` prior to releasing the mouse button.
- Using the Proof menu: Select Draw Box.... This will open a dialog box. In this dialog box, enter the start row and the end row of the box. The rows numbers correspond to the line numbers in the proof. The start and end rows can be the same to denote a box that encloses a single row. The start row can also be either less than or greater than the end row. The smaller of the two numbers will be used as the first line in the box.
- Using the toolbar: Click the button shown in Figure 1, then proceed in the same way as if using the Proof menu.

3.4.3 Resizing a box

To resize a box, first select the box by clicking on any of its edges. The box will now be shown with thicker lines colored red, and containing "handles" at the four corners and the midpoints of the two horizontal edges (see Figure 5). Next, move the mouse over any of the handles. When the mouse cursor changes to a resize icon, click and drag the selected edge to set its new position.

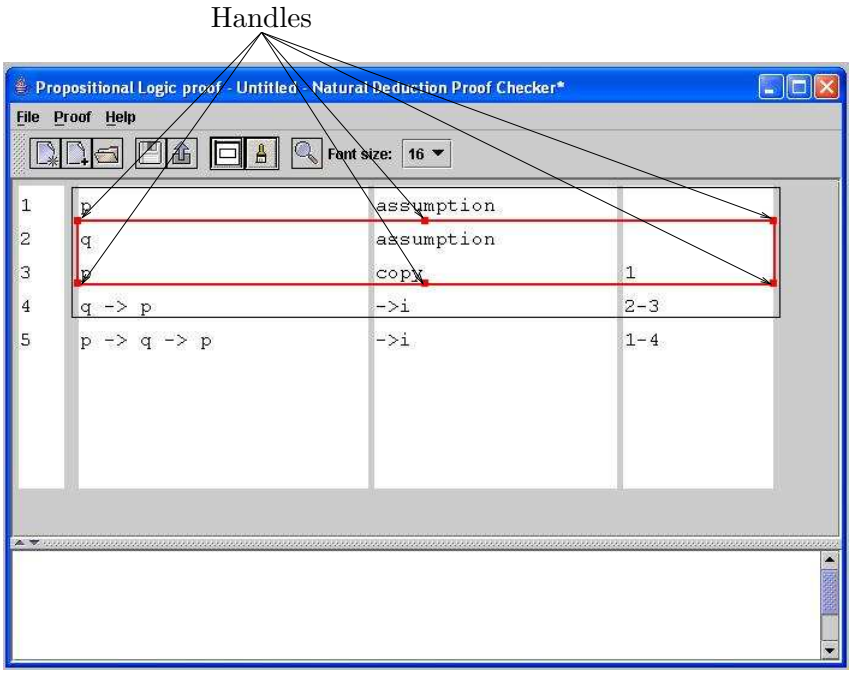

Figure 5: A selected box.

3.4.4 Deleting a box

To delete a box, first highlight the box by clicking on any of its edges, then press the `Delete` key.

3.5 Changing the Font Size

You can change the font size for the proof area using the drop-down menu labeled Font size: in the toolbar.

3.6 Checking a Proof

The correctness of a proof can be checked in any of the following ways:

- Using the Proof menu: Select Check.
- Using pop-up menu: Right-click in the proof area, then select Check Proof.
- Using toolbar: Click the button shown in Figure 1.
- Using the function key `F5`.

If an error is found, a message describing the error is displayed in the message area, and checking is terminated. If no errors are found, the message

```
Proof: OK
```

is displayed in the message area. This message indicates that the proof area contains a correct proof of the sequent

$$\phi_1, \ldots, \phi_n \vdash \psi$$

where $\phi_1, \ldots, \phi_n$ are all of the formulas in the proof having a justification of `premise`, and ψ is the last formula in the proof. For example, if you check the proof if Figure 4, no errors will occur; hence, it is a valid proof of

$$\texttt{~P(b)} \vdash \forall\texttt{x(x = b -> ~P(x))}$$

See Sections 4 and 5 for more details on the construction of correct proofs.

3.7 Clearing the Message Area

You can clear the message area in any of the following ways:

- Using the Proof menu: Select Clear Message Area.
- Using pop-up menu: Right-click in the proof area, then select Clear Message Area.
- Using toolbar: Click the button shown in Figure 1.
- Using the function key `F3`.

3.8 Saving a Proof

The primary ways of saving a proof to a file are as follows:

- Using the File menu: Select Save.
- Using toolbar: Click the button shown in Figure 1.
- Using the shortcut key `Ctrl+S`.

In addition, whenever an action would cause changes to a proof to be lost (e.g., starting a new proof or closing the program with an unsaved proof in the proof area), a confirmation dialog asks whether that proof should be saved. Clicking Yes also causes the proof to be saved using the same mechanism.

If any of the above occur, the manner in which the proof is saved depends upon whether the proof is associated with a file. If it is a new proof that has never been saved, a file browser will be displayed to allow the selection of a file name for the proof. If that file already exists, a warning to that effect will be displayed. If the Continue button is clicked, the proof will be saved in that file, overwriting the proof already stored there; otherwise the file browser will remain displayed so that a different file name can be selected. If Cancel is selected from the file browser, or if the file browser is closed, the proof will not be saved.

If the proof to be saved is already associated with a file (i.e., because it has been opened from that file or previously saved to that file), no file browser will be displayed — the proof will simply be saved to the associated file.

This behavior may be overridden, forcing a file browser to be displayed, using either of the following:

- Using the File menu: Select Save As....
- Using the shortcut key `Ctrl+A`.

3.9 Exporting a Proof to LaTeX

LaTeX is a markup language for producing documents in a variety of data formats, including PDF, PostScript, and DVI. It is particularly adept at formatting mathematical text. For example, LaTeX was used to create this document. The Natural Deduction Proof Checker provides a facility for exporting a proof formatted using LaTeX. The exported proof is in a form suitable for inclusion into a larger LaTeX document. For more information on installing an using LaTeX, see the following web site:

http://www.latex-project.org/

A LaTeX export may be produced using any of the following:

- Using the File menu: Select Export to LaTeX....
- Using pop-up menu: Right-click in the proof area, then select Export to LaTeX....
- Using toolbar: Click the button shown in Figure 1.
- Using the shortcut key `Ctrl+E`.

If any of the above are done, a file browser is displayed for the purpose of obtaining a file name. This file browser operates in the same way as the file browser described in Section 3.8.

Any LaTeX file that includes the exported file must use the packages amsmath, amssymb, array, and hhline. For example, suppose a proof is exported to the file `proof.tex`. A minimal file for producing a document containing this proof is shown in Figure 6. Figure 7 shows the LaTeX-formated version of the predicate logic proof shown in Figure 4.

A side-effect of including an exported proof is to set the LaTeX length variable extrarowheight to 0 points.

```
\documentclass{article}
\usepackage{amsmath,amssymb,array,hhline}
\begin{document}
\input{proof}
\end{document}
```

Figure 6: A LaTeX driver for formatting an exported proof.

1	$\neg P(b)$	premise
2	$x0$	
3	$x0 = b$	assumption
4	$b = x0$	Sym 3
5	$\neg P(x0)$	=e 4, 1
6	$x0 = b \rightarrow \neg P(x0)$	→i 3-5
7	$\forall x(x = b \rightarrow \neg P(x))$	∀i 2-6

Figure 7: A proof formatted by LaTeX.

3.10 Opening an Existing Proof

If you have saved a proof in your file system (see Section 3.8), you may open it in one of the following ways:

- Using the File menu: Select Open....
- Using pop-up menu: Right-click in the proof area, then select Open....
- Using toolbar: Click the button shown in Figure 1.
- Using the shortcut key `Ctrl+O`.

In response to one of the above actions, a file browser will be displayed. Using this file browser, you may navigate to the file containing the proof, then click Open. If the proof area contains an unsaved proof, a confirmation dialog will ask whether you want to save it first. If you click Yes, the existing proof will first be saved (see Section 3.8). If you click Cancel or close the confirmation dialog, the file will not be opened, and the existing proof will

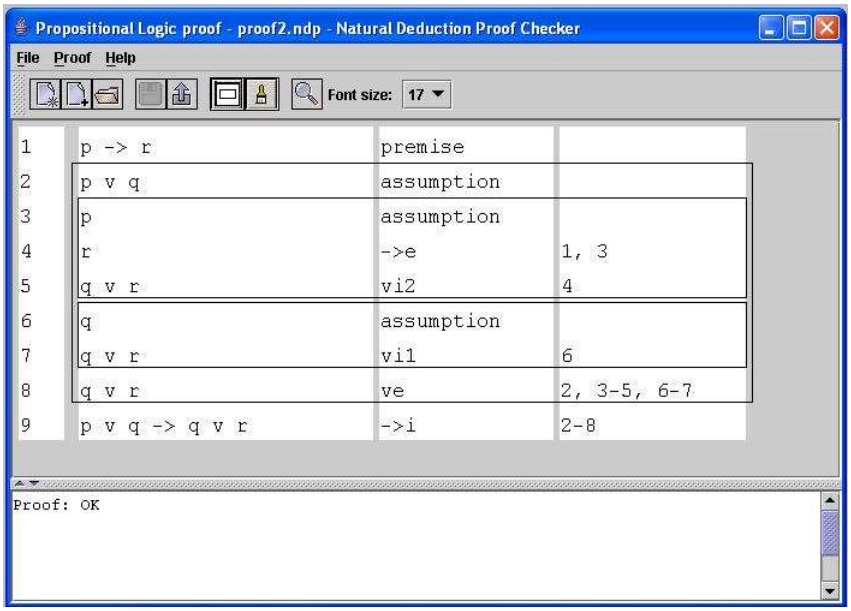

Figure 8: A proof of `p -> r` ⊢ `p v q -> q v r`.

not be saved; otherwise, the proof in the selected file will be displayed in the proof area.

3.11 Determining the Version Number

To determine the version number of your installation, select from the Help menu, About Natural Deduction Proof Checker. A window with the version number will open.

4 Propositional Logic Proofs

Figure 8 shows an example of a propositional logic proof in the proof checker. The proof consists of lines of text and boxes. Each line of text has four components, which correspond to the four columns in the proof checker:

- a line number, generated automatically;
- a propositional formula;

- a rule name; and
- a comma-separated list of 0 or more line numbers or ranges of line numbers (which should refer to boxes).

The line numbers listed in the fourth column refer to lines which are used to justify the given formula. For example, line 4 in Figure 8 contains the formula `r`, which is justified by lines 1 and 3 using the rule named `->e`. The boxes in the proof determine which lines may be used to justify a given formula.

In what follows, we will first describe what constitutes a valid propositional logic formula. We will then discuss how boxes determine which lines may be used to justify a given formula. Finally, we will present the proof rules and show how they can be used to construct proofs.

4.1 Propositional Logic Formulas

Propositional formulas are made up of propositional variables and logical connectives. The simplest formulas are propositional variables by themselves. A *propositional variable* is a string of alphanumeric characters of length 1 or more such that

- the first character is either `c`, `h`, `n`, `p`, `q`, `r`, `s`, or `t`; and
- no character is either `v` or `V` (which are reserved as operators).

Let ϕ and ψ denote propositional formulas. We can then construct the following propositional formulas:

Negation: `~`ϕ

Conjunction: ϕ `^` ψ

Disjunction: ϕ `v` ψ (or alternatively ϕ `V` ψ)

Implication: ϕ `->` ψ

Bi-implication: ϕ `<->` ψ

The multi-character connectives `->` and `<->` must be typed with no intervening blanks.

We use parentheses to denote the structure of a formula involving more that one connective; e.g., to connect the formulas `p ^ q` and `q -> r` with the connective `v`, we would write:

```
(p ^ q) v (q -> r)
```

In order to avoid an excess of parentheses, we introduce the following binding precedence among the connectives, from highest to lowest:

1. `~`
2. `^` and `v` (or equivalently `V`)
3. `->` and `<->`

In addition, all binary connectives (i.e., all connectives except `~`) associate to the right. Thus, for example

```
~p v q -> q ^ ~r ^ s
```

is equivalent to

```
((~p) v q) -> (q ^ ((~r) ^ s))
```

Note that formulas such as

```
p ^ q v r
```

or

```
p -> q <-> r
```

are ambiguous, and are therefore disallowed.

There is one final propositional formula representing a contradiction and denoted by the symbol $\bot$, which is typed using the underscore (_) key. Though it would make sense to combine $\bot$ with other formulas, the proof checker only allows $\bot$ to be used by itself.

4.2 Boxes and Accessibility of Proof Lines

The general rule for accessibility of proof lines or boxes is that the formula at line i may be justified by any preceding lines or boxes that are not contained in a box that does not also contain line i. Consider the proof in Figure 8, for example. Line 4 can use any preceding lines, because any box that contains a preceding line (e.g., line 3) also contains line 4. However, line 7 can use only lines 1, 2, and 6, as well as the box 3-5. Line 7 cannot use lines 3, 4, or 5, because they are contained in a box that does not contain line 7. Likewise, line 8 can use only lines 1 and 2 and boxes 3-5 and 6-7, and line 9 can use only line 1 and box 2-8.

4.3 Proof Rules for Propositional Logic

We begin our presentation of the proof rules for propositional logic by introducing three rules that do not rely on the form of the formulas they justify. These rules are as follows:

- `premise` — Any formula can be justified by this rule. It simply identifies the given formula as a premise for the proof. This rule does not use any line numbers in the fourth column.
- `assumption` — Again, any formula can be justified by this rule; however, whenever this rule is used, the line containing it must be the first line of a box. This rule also uses no line numbers in the fourth column.
- `copy` — Any accessible formula (see Section 4.2) can be copied using this rule. When this rule is used, the line number of the copied formula must be given in the fourth column.

For example, consider again the proof shown in Figure 8. First observe how the `premise` rule is used on line 1 to identify the premise `p -> r`. Also, note how the formulas on lines 2, 3, and 6 are justified by `assumption`, with each of these lines being the first line of a box. It is important that each rule be spelled correctly with no intervening blanks; however, case of the letters is not important. Thus, we could use `Premise` as the rule on line 1. Recall from Section 3.3, however, that typing an upper-case `A` or `E` produces the symbol ∀ or ∃, respectively; hence, using all caps to justify a premise would give `PR∃MIS∃` (which, in fact, is legal, but not very readable).

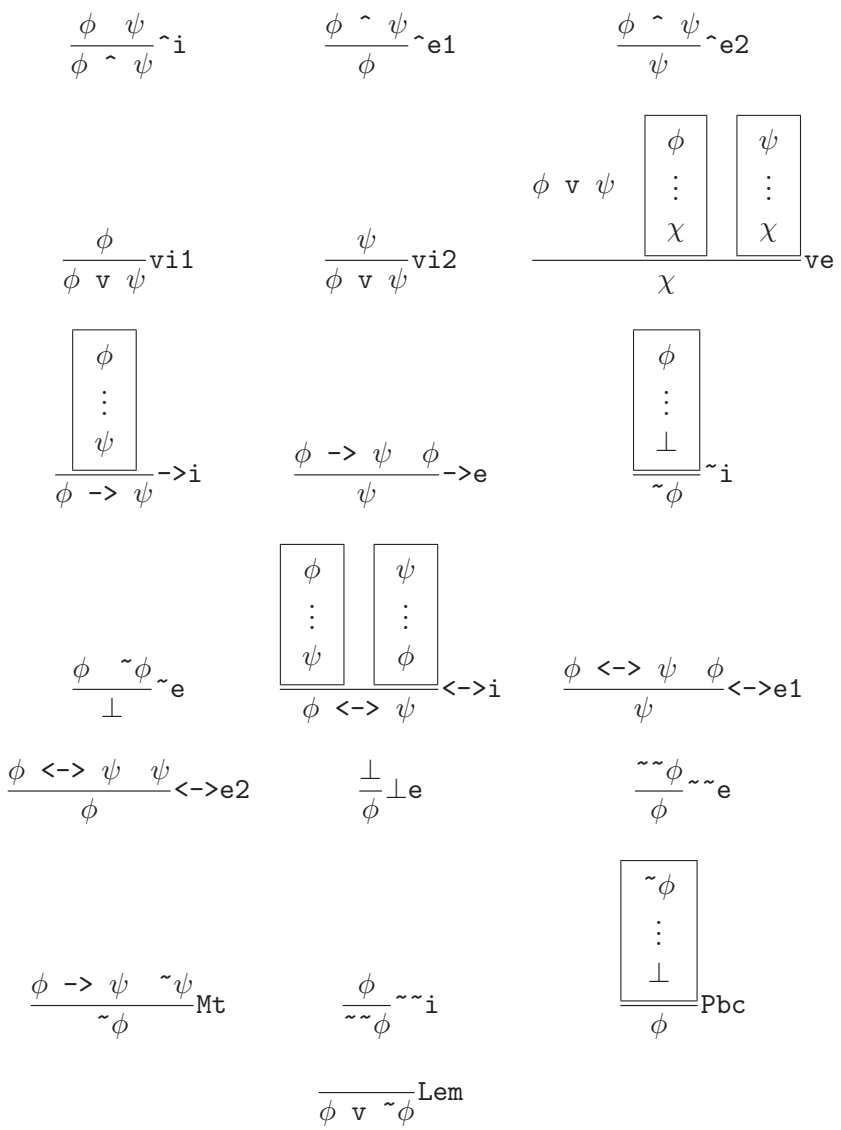

Figure 9: Proof rules for propositional logic.

The remaining proof rules are given in Figure 9. Each of these rules consists of three parts surrounding a horizontal line, as follows:

- Above the line are zero or more formulas and/or boxes representing the form of the *premises* of the rule.
- Below the line is a single formula representing the form of the *conclusion* of the rule.
- To the right of the line is a string representing the *name* of the rule.

Now consider line 4 of Figure 8. This line is derived using the proof rule named `->e` (pronounced "arrow elimination") — the name given in the third column.

Looking at Figure 9, we see that the `->e` rule has two premises above the line — ϕ `->` ψ and ϕ. The fourth column of line 4 must therefore contain a comma-separated list of two line numbers. These line numbers must reference accessible lines having the same form as the two premises of the rule. Line 1 contains the formula `p -> r`, which has the form of the first premise, ϕ `->` ψ. Because ϕ in the first premise matches `p` in line 1, the second premise, ϕ, must also match `p`. We see that the second line referenced in line 4 — namely, line 3 — does, in fact, contain the formula `p`. The premises of the rule therefore match the given line numbers in line 4.

Finally, the formula given in line 4 (`r`) must match the conclusion of the rule (ψ). Note that when we matched the first premise with line 1, ψ matched `r`. Therefore, the given formula does indeed match the conclusion of the rule. The rule has therefore been correctly applied to justify the formula on line 4. Using the same approach, we can easily see that the formulas on lines 5 and 7 are also properly justified.

Lines 8 and 9, however, are somewhat different in that they use boxes in their justifications. Let's consider line 9 first. It uses the rule named `->i` (pronounced "arrow introduction"). Referring again to Figure 9, we see that the `->i` rule has a single box as its premise. The top line of the box in this rule is ϕ, and the last line is ψ. This means that the box in the proof used by the rule must have as its first line a formula ϕ and as its last line a formula ψ. Note that neither of these lines may be further nested within boxes inside the box being referenced. Because ϕ and ψ do not have any restrictions on what they can match, we can match ϕ with `p v q` and ψ with `q v r`. The premise of the rule then matches the box 2-8.

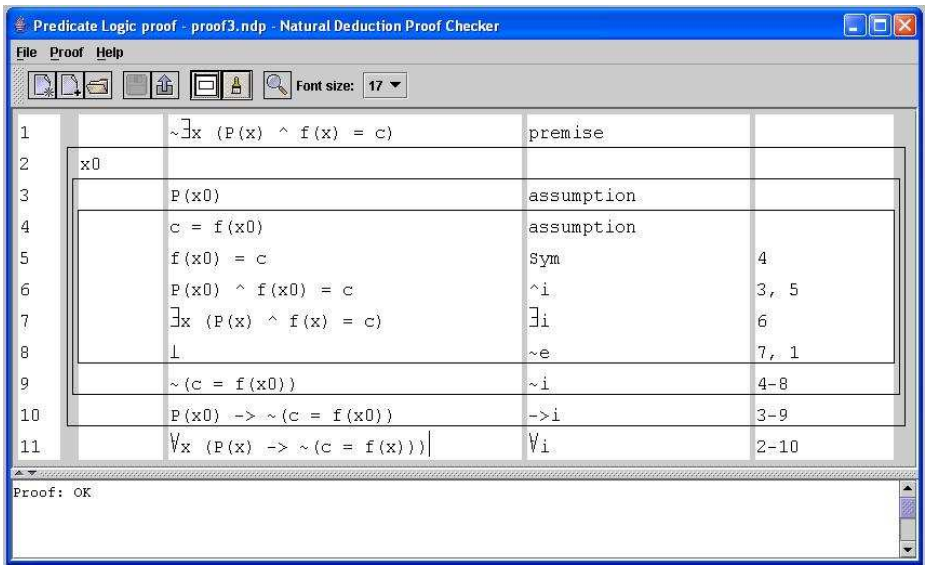

Figure 10: A predicate logic proof.

We now need to match the conclusion of the rule (i.e., ϕ `->` ψ) with the formula `p v q -> q v r`. Because we have already matched ϕ with `p v q` and ψ with `q v r`, we see that the conclusion does, in fact, match the formula. Similarly, we can see that the formula on line 8 is properly justified.

5 Predicate Logic Proofs

Figure 10 shows an example of a predicate logic proof in the proof checker. Many of the principles introduced in Section 4 carry over to predicate logic proofs. However, predicate logic proofs differ from propositional logic proofs in the following ways:

- Predicate logic proofs contain an extra column (the second column) for introducing temporary variables.
- Predicate logic formulas are more complex than propositional logic formulas.
- Predicate logic contains several new proof rules in addition to those of propositional logic.

In what follows, we will first describe what constitutes a valid predicate logic formula. We will then discuss how the introduction of variables in the second column restricts their use in a proof. Finally, we will present the proof rules and show how they can be used to construct proofs.

5.1 Predicate Logic Formulas

Predicate logic formulas are significantly more complex than propositional logic formulas. Instead of propositions, predicate logic formulas may contain constants, variables, function symbols, predicate symbols, and the special predicate symbol `=`. They also may contain two additional connectives. In what follows, we will define each of these components and how they are put together to produce formulas.

5.1.1 Constants and variables

The simplest components of predicate logic formulas are constants and variables. A *constant* is a string of alphanumeric characters of length 1 or more such that:

- the first character is either `c`, `d`, `e`, or `m`; and
- no character is `A`, `E`, `v`, or `V` (which are reserved as operators).

A *variable* is defined the same as a constant, except that the first character must be either `a`, `b`, `t`, `u`, `w`, `x`, `y`, or `z`.

5.1.2 Terms and functions

The simplest *terms* are either constants or variables. More complex terms can be constructed using *function symbols*, which are strings of alphanumeric characters of length 1 or more such that:

- the first character is either `f`, `g`, `h`, or `s`; and
- no character is `A`, `E`, `v`, or `V`.

Suppose f is a function symbol and $t_1, \ldots, t_n$ are terms, where $n \geq 1$. We can then construct the *function*

$$f(t_1, \ldots, t_n)$$

This function is then said to have *arity* n. Within any proof, functions containing the same function symbol must have the same arity. Each function is then a term.

The following are examples of functions, together with their arities:

function	arity
`f(x0)`	1
`g(f(x0), c)`	2
`h(g(f(x0)), d, f(c))`	3

5.1.3 Predicates

Predicates are formed from terms and *predicate symbols*. A predicate symbol is an alphanumeric string of length 1 or greater such that

- the first character is either `F`, `G`, `H`, `P`, `Q`, `R`, or `S`; and
- no character is `A`, `E`, `v`, or `V`.

Let P be a predicate symbol and $t_1, \ldots, t_n$ be terms, where $n \geq 1$. Then we can construct the following predicate:

$$P(t_1, \ldots, t_n)$$

This predicate is said to have *arity* n. In addition, we can construct a predicate P with arity 0. Within any proof, predicates containing the same predicate symbol must have the same arity.

The following are examples of predicates, together with their arities

predicate	arity
`P(x)`	1
`F(c, g(x0, y))`	2
`R`	0
`S(h(g(f(x0)), d, f(c)))`	1

In addition, we can construct predicates using the symbol `=`. Let t_1 and t_2 be terms. We can then construct the predicate

$$t_1 \texttt{ = } t_2$$

For example, from the terms `c` and `h(g(f(x0))`, we can construct the predicate

```
c = h(g(f(x0))
```

5.1.4 Formulas

We can now construct predicate logic formulas from predicates. The simplest formulas are just predicates. Let ϕ and ψ be predicate logic formulas and x be a variable. We can then construct the following predicate logic formulas:

Negation: `~`ϕ

Conjunction: ϕ `^` ψ

Disjunction: ϕ `v` ψ (or alternatively ϕ `V` ψ)

Implication: ϕ `->` ψ

Bi-implication: ϕ `<->` ψ

Universal quantification: $\forall x\phi$

Existential quantification: $\exists x\phi$

As for predicate logic, we use parentheses to denote the structure of a formula. To reduce the number of parentheses, we extend the binding precedence given in Section 4.1 as follows (highest to lowest):

1. `~`, $\forall$, and $\exists$
2. `^` and `v` (or equivalently `V`)
3. `->` and `<->`

As for propositional logic, all binary connectives associate to the right. Thus, for example,

```
~∃x∀y P(x) ^ Q(y) -> R(x, y)
```

is equivalent to

```
(~(∃x(∀y P(x))) ^ Q(y)) -> R(x, y)
```

Note that because `~`, $\forall$, and $\exists$ are all unary operators, there is never any ambiguity in precedence among them.

5.2 The Second Column

Certain of the proof rules for predicate logic (see Section 5.3) require that a variable be placed in the second column. There are two rules relevant to variables in this column.

- A variable may appear in the second column only as the first line of a box.
- If a variable appears in the second column, it may not appear anywhere outside the innermost box containing it.

Note that placing a variable in the second column requires no justification. In particular, it is legal to have a line containing only a variable in the second column and nothing in any of the following columns (see line 2 of the proof in Figure 10).

5.3 Proof Rules for Predicate Logic

All of the propositional logic rules shown in Figure 9 may be used in predicate logic proofs. In addition, the rules shown in Figure 11 may be used.

In order to be able to explain how these rules are applied, we must first explain the concepts of free variables and substitution.

5.3.1 Free occurrences of variables

Informally, a free occurrence of a variable is one that is not within the scope of a quantifier for that variable. In order to make this definition more precise, let ϕ and ψ denote predicate logic formulas, let $\cdot$ denote one of the binary connectives `^`, `v`, `V`, `->`, or `<->`, and let x and y denote distinct variables.

- If ϕ is a predicate, then all occurrences of x in ϕ are free.
- The free occurrences of x in `~`ϕ are the those occurrences that are free in ϕ.
- The free occurrences of x in $\phi \cdot \psi$ are those occurrences that are free in ϕ or ψ.

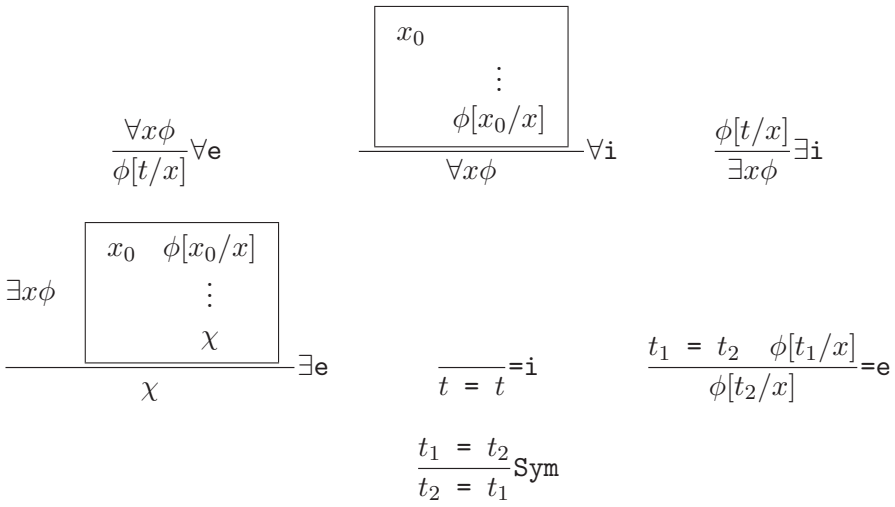

Figure 11: Additional proof rules for predicate logic.

- There are no free occurrences of x in $\forall x \phi$ or $\exists x \phi$; however, in both of these formulas, the free occurrences of y are those occurrences that are free in ϕ.

For example, in the formula

```
P(x) ^ ∀x (Q(x) -> P(y)) v ∃y R(x, y)
```

the first and last occurrences of `x` and the first occurrence of `y` are free, but the other occurrences are not.

5.3.2 Substitution

Let ϕ denote a predicate logic formula, t denote a term, and x denote a variable. We then use the notation $\phi[t/x]$ to denote the formula that results when we replace every free occurrence of x in ϕ by t. We call this notation a *substitution*. In order for a substitution to be valid, however, we add the additional restriction that every free occurrence of any variable in t must also be a free occurrence in $\phi[t/x]$. Thus, for example, substituting `f(x)` for `y` in `∀x P(x, y)` is invalid because the free occurrence of `x` in `f(x)` is no longer free in `∀x P(x, f(x))`.

5.3.3 Applying proof rules in predicate logic

The proof rules given in Figure 9 are applied as described in Section 4. The proof rules given in Figure 11 are applied in a similar way, but with more complicated pattern matching.

In Figure 11, ϕ and χ represent predicate logic formulas, x and x_0 represent variables, and t, t_1, and t_2 represent terms. Consider first the rule `∃i`. We apply this rule in order to derive a formula of the form $\exists x\phi$. In order to be able to apply this rule, we need a formula $\phi[t/x]$ — i.e., formula ϕ with some term t replacing every free occurrence of x, where every free occurrence of any variable in t is also free in $\phi[t/x]$. The form $\exists x\phi$ matches the formula

```
∃x (P(x) ^ f(x) = c)
```

on line 7 of the proof in Figure 10. In this case, ϕ matches

```
P(x) ^ f(x) = c
```

and x matches `x`. Furthermore, the formula on line 6 is just ϕ with all free occurrences of `x` (i.e., both occurrences) replaced by the term `x0`. Note that the only free occurrence of a variable in `x0` is `x0` itself, which remains free everywhere it has been substituted into ϕ. Therefore, line 7 contains a valid application of the rule `∃i`.

Now consider the more complicated rule `∀i`. This rule is used to derive a formula of the form $\forall x\phi$. The formula on line 11 of Figure 10 matches this form with ϕ matching

```
P(x) -> ~(c = f(x))
```

and x matching `x`. This rule requires a box whose first line introduces a variable x_0 in the second column and whose last line is ϕ with all free occurrence of x replaced by x_0, where all of these substitutions leave x_0 free. We see that the first line of the box 2-10 introduces the variable `x0` in the second column. Furthermore, the last line of this box is ϕ with both free occurrences of `x` replaced by `x0`. Finally, when this substitution is done, all substituted occurrences of `x0` remain free. Line 11 is therefore a valid application of `∀i`.

As a last example, consider the rule `=e`. An example of the use of this rule can be found on line 5 of the proof in Figure 4. Both the formula that is derived by this rule and one of the formulas required by this rule involve a

substitution for a variable x; however, x appears nowhere else in this rule. This means that x represents a variable that does not occur in the proof — it is simply a placeholder.

The easiest place to start when applying this rule is with the formula that must match t_1 `=` t_2. In Figure 4, this formula is on line 4: `b = x0`. Here, t_1 matches `b` and t_2 matches `x0`. The conclusion in line 5 and the other formula used (line 1) must both be derived by substitutions into some formula ϕ. In the former case, the substitution must be `x0` for x, and in the latter case, the substitution must be `b` for x. We can obtain this matching by letting ϕ represent `~P(`x`)`. We can then see that line 5 is a valid application of `=e`.

References

[1] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.